



Trail: The Reflection API

by Dale Green


The reflection API represents, or reflects, the classes, interfaces, and objects in the current Java Virtual Machine. You'll want to use the reflection API if you are writing development tools such as debuggers, class browsers, and GUI builders. With the reflection API you can:


- Determine the class of an object.
- Get information about a class's modifiers, fields, methods, constructors, and superclasses.
- Find out what constants and method declarations belong to an interface.
- Create an instance of a class whose name is not known until runtime.
- Get and set the value of an object's field, even if the field name is unknown to your program until runtime.
- Invoke a method on an object, even if the method is not known until runtime.
- Create a new array, whose size and component type are not known until runtime, and then modify the array's components.


First, a note of caution. Don't use the reflection API when other tools more natural to the Java programming language would suffice. For example, if you are in the habit of using function pointers in another language, you might be tempted to use the `Method` objects of the reflection API in the same way. Resist the temptation! Your program will be easier to debug and maintain if you don't use `Method` objects. Instead, you should define an interface, and then implement it in the classes that perform the needed action.


Other trails use the term "member variable" instead of "field." The two terms are synonymous. Because the `Field` class is part of the reflection API, this trail uses the term "field."

This trail uses a task-oriented approach to the reflection API. Each lesson includes a set of related tasks, and every task is explained, step by step, with a sample program. The lessons are as follows:

 **Examining Classes** explains how to determine the class of an object, and how to get information about classes and interfaces.

 **Manipulating Objects** shows you how to instantiate classes, get or set field values, and invoke methods. With the reflection API, you can perform these tasks even if the names of the classes, fields, and methods are unknown until runtime.

 **Working with Arrays** describes the APIs used to create and to modify arrays whose names are not known until runtime.

 **Summary of Classes** lists the classes that comprise the reflection API, and provides links to the appropriate API documentation.



[Start of Tutorial](#)

[Search](#)
[Feedback Form](#)

[Copyright](#) 1995-2003 Sun Microsystems, Inc. All rights reserved.



What's New in 1.1?

Object Serialization

Object Serialization extends the core Java Input/Output classes with support for objects. Object Serialization supports the encoding of objects and the objects reachable from them into a stream of bytes and the complementary reconstruction of the object graph from the stream. Serialization is used for lightweight persistence and for communication via sockets or Remote Method Invocation (RMI). The default encoding of objects protects private and transient data, and supports the evolution of the classes. A class may implement its own external encoding and is then solely responsible for the external format.

Where to Find Documentation

- The Object Serialization Specification can be found at [Object Serialization](#).
- In addition, you can look at the [java.io](#) API documentation.



What's New in 1.1?